An on-line approach to hybrid flow shop scheduling with jobs arriving over time

Verena Gondek, University of Duisburg-Essen

Abstract

During the manufacturing process in a steel mill, the chemical composition and quality of raw-iron, steel and slack has to be checked several times. The organization of the corresponding list of operations can be classified as a hybrid flow shop scheduling problem with jobs arriving over time and the objective is to minimize the total weighted completion time. Since this problem is known to be NP-hard, we deal with heuristic solution techniques to accomplish the required on-line optimization. To this end, we develop and evaluate a hybrid approach based on dispatching rules and bottleneck related strategies.

Keywords: hybrid flow shop scheduling, jobs arriving over time, total weighted completion time, heuristics, nwt-constraints

1 Introduction

(Hybrid) flow shop scheduling problems (HFS) occur in various real world applications. Within a HFS problem a set of n jobs has to be processed sequentially on m different machine types where n_i machines of type i are available in parallel. Each machine can handle at most one job at a time and each job can be assigned at most to one machine at a time. Job preemptions are not allowed. In this paper, we deal with a real-life problem arising in steel producing industries. For monitoring the manufacture of steel in a big German steel mill, product samples are taken at several stages of the production process and are sent to an automatic laboratory to check their quality. Therefore, they have to be processed on different machines. The upcoming transportation tasks between the operations are performed by a fleet of robots. The efficient organization of the workflow in this laboratory can be classified as a hybrid flow shop scheduling problem with transportation requests and jobs arriving over time. That is, the jobs' arrival times are not known in advance, but all information about its processing requirements becomes known when a job is released.

We decided to decompose the problem into two subproblems and to treat them consecutively, because the additional constraints related to the 'normal' machines are completely different from those associated with the robots. Thus, in a first step we allocate the available machines to the set of samples and determine the corresponding job sequence for each machine. We refer to that subproblem as the *sequencing problem*. Based on its solution, we determine a schedule for the resulting transportation requests in a second step, which we call the *routing problem*. In this paper, we just concentrate on the former. Due to varying job priorities and since there is no intermediate buffer between the different production stages available in the laboratory, the sequencing problem can be classified as a $FF|(r_j), nwt|\sum w_jC_j$-problem with jobs arriving over time, according to the three-field classification scheme introduced by Graham et. al. (1979) [8].

During the last decades, hybrid flow shop scheduling has received increasing attention. Nevertheless, the majority of flow shop related research is either focused on minimizing makespan or restricted to two-staged problems (cf. Linn & Zhang, 1999, [13]; Liu et. al., 2005, [14]). The same applies to the corresponding on-line versions, especially with jobs arriving over time (e.g. Sgall, 1998, [22]; Pruhs et. al., 2004, [19]). Except for the work of Vestjens (1997) [23], we are not aware of any approach to solving (hybrid) flow shop scheduling problems with jobs arriving over time in the literature, but Vestjens considers exclusively makespan minimization, too.

Just like most kinds of scheduling tasks, (hybrid) flow shop problems are well-known to be NP-hard, even if there are no additional constraints considered (e.g. Pinedo, 2008, [18]). Due to the lack of computational time within our practical context, we developed a fast heuristic approach for solving the $FF|(r_j), nwt|\sum w_jC_j$-problem with jobs arriving over time. The hybrid on-line method is based on the well-known WSPT dispatching rule, bottleneck related strategies as well as job-driven list scheduling and adapts a general technique for the construction of on-line scheduling methods suggested by Hall et. al. (1997) [9] and Phillips et. al. (1998) [17]. According to the practical application at hand, we actually deal with no-waiting-time constraints, but our algorithm can be adapted easily to the case of unlimited intermediate buffer. Thus, the main contribution of

this paper is a new and fast heuristic to solve the HFS problem with total weighted completion time objective and jobs arriving over time.

The remainder of this paper is organized as follows: In Section 2 we give a formulation of the problem as a mixed-integer program. A brief overview on related work in the literature and our new heuristic approach are presented in Section 3. Section 4 is dedicated to a comprehensive computational study. Finally, some concluding remarks are given in Section 5.

2 Problem formulation

In this section, we present a mixed-integer formulation for the considered hybrid flow shop scheduling problem with jobs arriving over time and no waiting time constraints, which is based on approaches by Dyer and Wolsey (1990) [5] and Gomes et. al. (2005) [6]. Remember that n jobs have to be sequentially processed on m different machine types where n_i identical machines of type i are available in parallel. Thus, we can constitute the problem as a standard flow shop problem with a capacity of n_i jobs per time for each machine i. Before introducing the model, we define the parameters and decision variables used.

2.1 Parameters

Let us denote by

$J = \{1, \ldots, n\}$ - the set of jobs

$I = \{1, \ldots m\}$ - the ordered set of processing stages, each job has to pass through the stages in the same order $1 \rightarrow \ldots \rightarrow m$

n_i - the number of available machines on a stage $i \in I$

r_j - the release date of job $j \in J$

w_j - the weight of job $j \in J$

p_{ij} - the processing time of job $j \in J$ on stage $i \in I$ (the machines on the same stage are identical)

$t = 0, \ldots, T$ - the scheduling horizon (T can be any adequate upper bound on the makespan)

2.2 Decision variables

Two types of decision variables are used in the formulation:

$x_{ijt} \in \{0, 1\}$ is a binary variable to decide, if a job is processed on a stage in a time interval or not, more precisely:

$$x_{ijt} = \begin{cases} 1 & \text{, if job } j \text{ is processed on machine } i \text{ in the time interval } [t, t+1] \\ 0 & \text{, otherwise} \end{cases}$$

$C_{ij} \geq 0$ denotes the completion time of job j on stage i.

2.3 The model

Using the above symbols, the off-line version of the sequencing problem can be formulated as follows:

$$\text{Minimize } \sum_{j=1}^{n} w_j C_{mj} \tag{1}$$

subject to

$$\sum_{j=1}^{n} x_{ijt} \leq n_i \qquad \text{for } i = 1,\ldots,m;\ t = 0,\ldots,T \tag{2}$$

$$\sum_{i=1}^{m} x_{ijt} \leq 1 \qquad \text{for } j = 1,\ldots,n;\ t = 0,\ldots,T \tag{3}$$

$$\sum_{t=0}^{T} x_{ijt} = p_{ij} \qquad \text{for } i = 1,\ldots,m;\ j = 1,\ldots,n \tag{4}$$

$$C_{(i+1)j} - C_{ij} = p_{(i+1)j} \qquad \text{for } i = 1,\ldots,m-1;\ j = 1,\ldots,n \tag{5}$$

$$C_{1j} - p_{1j} \geq r_j \qquad \text{for } j = 1,\ldots,n \tag{6}$$

$$C_{ij} = \frac{p_{ij}+1}{2} + \frac{1}{p_{ij}} \sum_{t=0}^{T} t \cdot x_{ijt} \qquad \text{for } j = 1,\ldots,n \tag{7}$$

$$x_{ijt} = 0 \qquad \text{for } j = 1,\ldots,n;\ i = 1,\ldots,m;\ t = 0,\ldots,T;\ t < r_j \tag{8}$$

The objective (1) is to minimize the total weighted completion time. Conditions (2) restrict the number of jobs processed in parallel at the same time to the number of available machines on each stage. Job splitting is prevented by constraints (3). Conditions (4) impose the total time requirements of each operation. Constraints (5) define the operation precedence among the stages for the jobs. Note that if we replace '=' by '$\geq$' in this equation, the model will correspond to the situation with unlimited intermediate buffer. Further, (6) ensures that the processing of a job starts after its release. Finally, conditions (7) and (8) define the interdependence between the two types of decision variables. This formulation of the relationship was originally introduced by Dyer and Wolsey (1990) [5] for the $1|r_j|\sum w_j C_j$-problem. However, the proof of the correctness can be adapted to our case: Consider a job j is processed on stage i in a time interval $[t_0, t_0 + p_{ij}]$. Thus, we have $x_{ijt} = 1$ for $t = t_0,\ldots,t_0 + p_{ij} - 1$ and $C_{ij} = t_0 + p_{ij}$. Plugging in these values in equation (7) leads to

$$\begin{aligned} C_{ij} = \frac{p_{ij}+1}{2} + \frac{1}{p_{ij}} \sum_{t=0}^{T} t \cdot x_{ijt} &= \frac{p_{ij}+1}{2} + \frac{1}{p_{ij}} \left(t_0 + (t_0+1) + \ldots + (t_0 + p_{ij} - 1)\right) \\ &= \frac{p_{ij}+1}{2} + \frac{1}{p_{ij}} \left(t_0 p_{ij} + \frac{p_{ij}(p_{ij}-1)}{2}\right) \\ &= t_0 + p_{ij} \end{aligned}$$

Equation (8) is necessary to guarantee the feasibility of the binary variables with respect to the release dates. To reach feasibility for all purposes, we would have to add another class of constraints imposing that each job is processed during p_{ij} consecutive time intervals on a stage i. This aspect is already implicated for all stages i with $i \neq 1$, in terms of the no-waiting-time constraints (5). However, the aim of the model above is not to compute exact solutions for the problem. Within our computational study in Section 4 we utilize the lp relaxation of the model to obtain a lower bound for the considered problem instances to evaluate the quality of our heuristic approach presented in the next section. Therefore, we refrain from incorporating the missing class of constraints for the first production stage.

The allocation of the jobs to the machines is not given by a solution of the program above. We assumed identical machines in parallel at each stage, hence, the assignment can either result directly from the solution or can be determined arbitrarily if it is not unique. But even if we had non-identical machines, we could use the problem's solution (respectively an adequate approximation) as the first stage of a decomposition method and treat the assignment of the jobs to the individual machines as another subproblem. Therefore, a prioritization of the machines, according to different speeds combined with a greedy-fashioned algorithm is imaginable. However, within this article we restrict ourselves to the case of identical machines.

3 A hybrid list scheduling algorithm

List scheduling strategies are the origin of manifold scheduling heuristics. They all share the same basic concept: A priority list of the jobs is determined, for instance, by any convenient dispatching rule and afterwards the jobs are scheduled one by one in the given order. The scheduling decision in that second step mostly depends on the partial solution provided by previously scheduled jobs (cf. Hurink & Knust, 2001, [10]). Thereby, an elementary distinction is made between machine-based and job-driven strategies. The former is also known as Graham's non-idling list scheduling strategy (cf. Graham, 1969, [7]), that is, every time a machine is freed the first job in the priority list is selected and assigned to that machine. As pointed out by Queyranne and Schulz (2006) [20] such an approach is appropriate for the optimization of throughput related objectives, such as makespan minimization, whereas they are inapplicable for job-oriented objectives, like the total weighted completion time. Thus, the dynamic algorithm, which is presented in the following, is based on a job-driven list scheduling strategy.

Due to the strict limitation of computational time, we are convinced that the adaptation of suitable dispatching rules in combination with list scheduling is the only possibility to comply with the present runtime requirements. Therefore, we develop a hybrid algorithm, which exploits the advantages of different fast dispatching rules. In the following paragraph, we introduce our dynamic list scheduling approach first, and afterwards we present the different sequencing methods that can be used within this algorithm.

3.1 Dynamic job-driven list scheduling

We use the following additional notations to describe our algorithm.

- $J(t)$ - set of jobs j, which are already released at time t, i.e. $r_j \leq t$.
- $L(t)$ - set of jobs $j \in J(t)$, which are released but not in process at time t yet.
- $cap(i,t) \leq n_i$ - workload of machine stage i at time t. Additionally, we use a parameter $cap'(i,t)$ in the same manner, to model the temporal workload of the machines during the application of the list scheduling approach to the different job sequences.
- s_{ij} - start time of job j on the stage i. The start times on the stages i with $i > 1$ follow implicitly from s_{1j}, according to the no waiting time constraints.

Recall that we consider an on-line scheduling problem with jobs arriving over time, thus, all information about the job's processing requirements becomes known at its release time. According to the methods proposed by Hall et. al. (1997) [9] and Phillips et. al. (1998) [17], we reoptimize the whole system, whenever a new job is released and therefore, we adapt methods originally designed for off-line scheduling (see also Section 3.2). Within such a reoptimization step at time t we just consider the jobs contained in $L(t)$ and do not try to utilize any future forecast. In each step we make use of a set of sequencing rules (denoted by H) and apply them to the jobs in $L(t)$. For each of the resulting job sequences we execute the job-driven list scheduling algorithm, which is presented in the following. Afterwards, we choose the job sequence with the best objective function value (minimal total weighted completion time).

Based on a given job sequence we schedule the jobs according to the following greedy list scheduling rule: Select the first job in the list and insert it into the current partial schedule such that its processing on the first machine is started as early as possible. Because we have no intermediate buffer, the start times on the other stages follow implicitly. This might cause idle time on some machines, even if jobs are already available, and therefore, we have a job-driven list scheduling algorithm. Every time t a job is released, we reschedule all jobs in $L(t)$. Obviously, jobs that are already in process cannot be rescheduled, but those scheduled and not started yet, can be involved in the reoptimization process. Hence, the dynamic list scheduling algorithm can be formalized as follows. At the beginning, when no job is scheduled, set $cap(i,t) = 0$ for all i,t. Every time a job j is released, do the following:

1. Set $t_0 := r_j$ and $s_j := T$.
2. Insert j into $J(t_0)$ and set $L(t_0) := \{k \in J(t_0) | r_k \leq t_0 \leq s_{1k}\}$.
3. For all jobs in $L(t_0)$:
   If $s_{1j} < T$
   Set $cap(i,t) = cap(i,t) - 1$ for $t = s_{ij}, \ldots, s_{ij} + p_{ij} - 1$ for all stages $i \in I$.
4. For each heuristic $h \in H$:

Apply h to $L(t_0)$. The resulting job sequence is denoted by $j_1^h, \ldots, j_r^h$.

For all jobs $j = j_1^h, \ldots, j_r^h$:

(a) Set $t_{j_k}^h := t_0$

(b) Set $cap'(i,t) := cap(i,t)$.

(c) If $cap'(i,t) = n_i$ for at least one stage i and one $t = t_{j_k} + \sum_{l=1}^{i-1} p_{lj_k}, \ldots, t_{j_k} + \sum_{l=1}^{i} p_{lj_k} - 1$

Set $t_{j_k}^h := t_{j_k}^h + 1$ and check the above condition again

Otherwise

Set $s_{1j_k}^h := t_{j_k}^h$ and $s_{ij_k}^h := s_{i-1j_k}^h + p_{i-1j_k}$ for $i = 2, \ldots, m$.

Set $C_{j_k}^h = s_{1j_k}^h + \sum_{i=1}^{m} p_{ij_k}$.

Set $cap'(i,t) = cap'(i,t) + 1$ for $i = 1, \ldots, m$ and $t = s_{ij_k}, \ldots, s_{ij_k} + p_{ij_k} - 1$.

(d) Compute the resulting objective function $z^h := \sum_{j \in L(t_0)} w_j C_j^h$.

5. Choose the job sequence $j_1^{h^*}, \ldots, j_r^{h^*}$ with

$$z^{h^*} := \min_{h \in H} z^h.$$

6. For all jobs in $L(t_0)$:

Set $s_{1j_1} := s_{1j}^{h^*}$ and $s_{ij} := s_{ij}^{h^*}$.

Set $cap(i,t) = cap(i,t) + 1$ for $i = 1, \ldots, m$ and $t = s_{ij}, \ldots, s_{ij} + p_{ij} - 1$.

In the first three steps, the system is prepared for reoptimization. Thus, the two sets $J(t_0)$ and $L(t_0)$ are updated according to the present point in time t_0. All jobs that are already assigned to a time slot on each processing stage, but will be rescheduled now, have to be removed from the actual schedule and therefore, we have to decrease the workload of the particular machines (see Step 3). In Step 4, the different sequencing methods are applied to the jobs in $L(t_0)$. Obviously, $t_{j_k} := t_0$ is the earliest possible start time for each job j_k, hence, we first check, if j_k can be started at time $t_{j_k} = t_0$. If this is not possible, we increment t_{j_k} as long as a valid start time for job j_k is found. After application of all heuristics $h \in H$, we choose the job sequence leading to the smallest objective function value and update the start times of the jobs as well as the machine workload parameters accordingly (Steps 5 and 6).

3.2 Sequencing methods

In the following, we specify the set of sequencing heuristics H used in the algorithm described above. Therefore, we give a brief overview on the impact of several dispatching rules and heuristics on flow shop related research, which motivated our approach. For example, Schulz (1996) [21] uses an lp relaxation to determine an initial job sequence for the application of list scheduling. In doing so, he is able to prove a worst case performance guarantee of $3m$ for the $FFm||\sum w_j C_j$-problem as well as a bound of $3m + 1$ for the same problem with release dates ($FFm|r_j|\sum w_j C_j$). To the best of our knowledge, these are the first known results about the performance of approximation algorithms in hybrid flow shop scheduling. Nevertheless, the approach is not really applicable to real world scheduling problems, because of the necessary computational effort to solve the suggested lp relaxation.

Fortunately, simple dispatching rules can be useful within flexible flow shop scheduling, too. For instance, Rajendran and Chaudhuri (1992) apply the well-known shortest processing time first rule (SPT) to sequence jobs and to tackle the problem $FF2||\sum C_j$. In their computational study, the average gap between approximative result and optimal solution is about 2%. Azizoglu et. al. (2001) [1] refer to that paper and utilize the SPT rule to solve instances of the problem $FFm||\sum C_j$. In relation to the total weighted completion time objective, Kyparisis and Koulamas (2001) [12] are successful in adapting the weighted shortest processing time first (WSPT) rule, also known as Smith's rule, which provides an optimal solution to the problem $1||\sum w_j C_j$ (e.g. Pinedo, 2008, [18]). In doing so, they yield a worst case performance bound of $(1+\sqrt{2})/2 \cdot \lceil \sum n_m / \min n_m \rceil$ for the problem $FFm||\sum w_j C_j$.

In the current literature, only asymptotic analysis provides a few further findings. For instance, Kaminsky et. al. (1998) also use the WSPT rule with respect to the total processing time of each job (WSTP) and prove that this algorithm is asymptotically optimal for the standard flow shop scheduling problem with a single machine on each stage ($Fm||\sum w_j C_j$). A similar strategy is applied by Liu et. al. (2005) [14] who show the asymptotic optimality of a WSPT based algorithm for the problem with release dates ($Fm|r_j|\sum w_j C_j$).

All mentioned heuristics are made for off-line situations, but as mentioned before, we intend to utilize successful off-line strategies for our on-line method. For instance, Megow and Schulz (2004) show that this approach might be promising. They work on the on-line version of the $Pm|r_j|\sum w_jC_j$-problem. Once more, the WSPT rule is effectively adapted. Another example is the work of Chakrabarti et. al. (1996)[2] who utilize the SPT rule and yield a 2-competitive algorithm for the problem $Pm|r_j, prmp|\sum C_j$ and respectively, a 6-competitive strategy for the problem without preemptions. For further results, we refer the reader to the comprehensive overviews on on-line scheduling algorithms by Sgall (1998) [22] and Pruhs et. al. (2004) [19].

Another central idea within the development of adequate methods for job sequencing is the following observation: In a shop scheduling problem, there are often considerable differences in the workload of the various production stages, i.e., there may be one or more bottleneck stages who have a great impact on the performance of the whole system (cf. Azizoglu et. al., 2001, [1]). Nevertheless, such aspects are not involved in any of the approaches mentioned before. Probably, the most popular bottleneck related scheduling algorithm is the so-called shifting bottleneck heuristic (cf. Pinedo, 2008, [18]), which solves scheduling problems of job shop type with makespan minimization. The performance of this procedure is promising, but unfortunately, the determination of the bottleneck stage requires the solution of various NP-hard single machine problems. Because of the restricted computational time in our special case, other methods should be explored. For instance, Paternina-Arboleda et. al. (2007) [15] deal with a very natural bottleneck definition. They identify the bottleneck stage as the stage with maximum relative workload (i.e., the total workload divided by the number of available machines). This strategy is much faster and moreover, the results are comparable to those obtained by the shifting bottleneck heuristic.

In the following, we present the five different sequencing methods we use within our list scheduling algorithm and that are motivated by the known results summarized above.

1. **WSTP sequence**

   (1) The jobs are sequenced in increasing order according to their weighted total processing time, that is, we have the sequence $j_1, j_2, \ldots, j_n$, if

   $$\frac{\sum_{i\in I} p_{ij_1}}{w_{j_1}} \leq \frac{\sum_{i\in I} p_{ij_2}}{w_{j_2}} \leq \ldots \leq \frac{\sum_{i\in I} p_{ij_n}}{w_{j_n}}.$$

2. **Bottleneck-related sequences**

   According to the approach by Paternina-Arboleda et. al. (2007) [15], we define the production stage with the biggest workload in relation to the number of available machines as the bottleneck of the system. That means, within the stages $i \in I$, we select the bottleneck i^* subject to

   $$w(i^*) = \max_{i\in I} w(i), \text{ with } w(i) := \frac{\sum_{j\in L(t)} p_{ij}}{n_i} \text{ for all } i \in I$$

   There are several opportunities to schedule the bottleneck stage i^* and to schedule the others as well. At first, we interpret the scheduling of the machines of type i^* as a scheduling problem with identical machines in parallel, no additional restrictions and total weighted completion time objective ($Pm||\sum w_jC_j$). Afterwards, a corresponding optimal or approximative job sequence for this problem constitutes an initial job list for the list scheduling algorithm. To approximate the problem $Pm||\sum w_jC_j$ we make use of the WSPT rule[1] once more, and in this way we obtain the following sequencing rule.

   (2.1) The jobs are sequenced in increasing order according to their weighted processing time on the bottleneck stage i^*, that is, we have the job sequence $j_1, j_2, \ldots, j_n$, if

   $$\frac{p_{i^*j_1}}{w_{j_1}} \leq \frac{p_{i^*j_2}}{w_{j_2}} \leq \ldots \leq \frac{p_{i^*j_n}}{w_{j_n}}.$$

   According to the no waiting time constraints, we have no flexibility in scheduling the other stages i with $i \neq i^*$. Hence, it might be senseful to involve available information about the other stages in the scheduling process for the bottleneck stage. Once a job is released, its processing requirements on all production stages are known, thus, we use the processing times of a job on preceding stages to compute its earliest possible start time at the bottleneck stage. More precisely, we define:

   $$r_{i^*j} := \sum_{h=1}^{i^*} p_{(h-1)j}, \text{ whereby } p_{0j} := \max\{t_0, r_j\} \text{ for all } j \in J.$$

[1] Kawaguchi and Kyan (1986) [11] showed that the WSPT dispatching rule is a $((1+\sqrt{2})/2)$-approximation for $Pm||\sum w_jC_j$.

Consequently, we have to solve the problem $Pm|r_j|\sum w_jC_j$, which is known to be NP-hard (cf. Chekuri & Khanna, 2004, [3]) for the bottleneck stage i^*. Therefore, we deal with the preemptive version $Pm|r_j, pmnt|\sum w_jC_j$ of the problem and approximate it by the preemptive version of the WSPT rule (note that preemptions are only allowed at discrete points in time). Afterwards, there are different possibilities for sequencing the jobs. Among others, we adapt the conversion algorithm provided by Phillips et. al. (1995,1998) [16, 17] to transform a preemptive schedule into a feasible one without preemptions, in which jobs are sequenced in the order of their completion times in the preemptive schedule. We use the resulting sequence for the bottleneck stage as a job sequence for our list scheduling approach.

As pointed out by Queyranne and Schulz (2006) [20], sequencing of the jobs according to their completion times in a relaxation of a problem may lead to very poor results for total (weighted) completion time minimization, because jobs with small processing times are treated in the same way as jobs with long processing times. Therefore, the authors deal with the completion midpoints $M_j^* := C_j^* - \frac{1}{2}p_{i^*j}$ as coefficients to sequence the jobs. In addition to that method we consider an α-scheduler (with $\alpha = \frac{1}{2}$, see Chekuri et. al., 2001, [4]) to involve the processing requirements of the jobs in the decision making process. Altogether, we utilize the following three sequencing heuristics:

(2.2) We have the job sequence $j_1, \ldots, j_n$, if

$$C_{j_1}^* \leq C_{j_2}^* \leq \ldots \leq C_{j_n}^*.$$

(2.3) Define $M_j^* := C_j^* - \frac{1}{2}p_{i^*j}$. Then we have the job sequence $j_1, \ldots, j_n$, if

$$M_{j_1}^* \leq M_{j_2}^* \leq \ldots \leq M_{j_n}^*.$$

(2.4) Define $C_j^*(\alpha)$ as the point in time when a fraction of αp_{i^*j} of the processing requirement p_{i^*j} is completed, with $0 < \alpha \leq 1$. We deal with $\alpha = \frac{1}{2}$ and therefore, we have the job sequence $j_1, \ldots, j_n$, if

$$C_{j_1}^*(\tfrac{1}{2}) \leq C_{j_2}^*(\tfrac{1}{2}) \leq \ldots \leq C_{j_n}^*(\tfrac{1}{2}).$$

Note that (2.2) is equivalent to (2.4) with $\alpha = 1$.

In the described list scheduling algorithm, these five sequencing methods constitute the set H. If $L(t)$ contains at most three jobs, it will not make sense to use all the sequencing methods. Instead, it is advisable to deal with all (at most six) possible job sequences, because the effort is nearly the same and we definitely find the best job sequence.

4 Computational Study

The aim of our computational study is to analyze the behavior of the different sequencing strategies as well as the performance of the whole method. The algorithm was implemented in C++ and furthermore we used Ilog OPL Studio 3.7 to obtain optimal solutions for small problem instances and to solve the lp relaxation of the presented mixed-integer program for bigger instances. All instances were solved by using a 1.7 MHz Pentium M processor and 1 GB RAM available. The different data instances were generated as follows.

As a first test series (data set 1), we randomly generated a set of 140 data instances with 5,10,15 or 20 jobs. Furthermore, we considered 2,3,5,7 or 10 different production stages. Of course, the number of jobs is no that big, but it is comparable to our practical situation, because at most 15 samples can be received in the laboratory at the same time, due to physical restrictions. In practice, there are usually no more than eight samples arriving simultaneously. The values for job processing times, release dates, weights as well as the machine capacities were randomly drawn as integer values from a uniform distribution using the following intervals:

- $p_j \in [1, 10]$ for all $j \in J$
- $r_j \in [1, R]$ for all $j \in J$, R depends on the number of jobs ($R = 20, 30, 40$ or 50), because an instance with only one or two jobs available at the same time (and therefore no application of the sequencing methods) would not be that representative
- $w_j \in [1, 5]$ for all $j \in J$
- $n_i \in [1, 5]$ for all $i \in I$

$n \times m$	(1)	(2.1)	(2.2)	(2.3)	(2.4)	best	on-line
5×2	0,0444	0,0482	0,0167	0,0167	0,0167	0,0167	0,0151
5×3	0,0424	0,0554	0,0426	0,0426	0,0426	0,0412	0,0746
10×2	0,0547	0,0589	0,0394	0,0397	0,0326	0,0310	0,0351
10×3	0,0500	0,0501	0,0445	0,0421	0,0382	0,0325	0,0333
10×5	0,1649	0,1554	0,1244	0,1241	0,1224	0,1170	0,1371
10×7	0,1202	0,1239	0,1153	0,1154	0,1064	0,0997	0,1160
15×2	0,0597	0,0519	0,0384	0,0370	0,0320	0,0313	0,0352
15×3	0,1340	0,1502	0,0994	0,0930	0,1013	0,0914	0,0846
15×5	0,1753	0,1649	0,1300	0,1271	0,1220	0,1185	0,1253
15×10	0,1012	0,1225	0,1178	0,1181	0,1180	0,0932	0,0978
20×2	0,1024	0,1069	0,0695	0,0610	0,0377	0,0373	0,0418
20×3	0,0789	0,0972	0,0753	0,0708	0,0653	0,0585	0,0613
20×5	0,1915	0,2508	0,2078	0,2013	0,2022	0,1711	0,1668
20×10	0,1324	0,1598	0,1277	0,1286	0,1296	0,1186	0,1211
total ($\emptyset$)	**0,1037**	**0,1140**	**0,0892**	**0,0870**	**0,0834**	**0,0759**	**0,0818**

Table 1: Relative error vs. lower bound (data set 1)

$n \times m$	(1)	(2.1)	(2.2)	(2.3)	(2.4)	best	on-line
5×2	0,0256	0,0252	0,0251	0,0187	0,0187	0,0187	0,0285
5×3	0,0309	0,0243	0,0150	0,0146	0,0131	0,0131	0,0159
10×2	0,0594	0,0374	0,0369	0,0347	0,0327	0,0294	0,0281
10×3	0,0650	0,0781	0,0610	0,0610	0,0630	0,0556	0,0562
10×5	0,0248	0,0339	0,0265	0,0264	0,0254	0,0209	0,0271
10×7	0,0422	0,0523	0,0266	0,0248	0,0248	0,0239	0,0390
15×2	0,0823	0,0940	0,0690	0,0641	0,0640	0,0578	0,0653
15×3	0,1128	0,0930	0,0859	0,0754	0,0691	0,0630	0,0802
15×5	0,1052	0,1287	0,1055	0,1033	0,0997	0,0775	0,0850
15×10	0,1012	0,1225	0,1178	0,1181	0,1180	0,0932	0,0978
20×2	0,1350	0,1365	0,0918	0,0819	0,0749	0,0719	0,0745
20×3	0,1256	0,1408	0,1113	0,0988	0,1075	0,0949	0,1140
20×5	0,1430	0,1662	0,1396	0,1386	0,1391	0,1280	0,1149
20×10	0,1431	0,1674	0,1643	0,1564	0,1612	0,1315	0,1331
total ($\emptyset$)	**0,0854**	**0,0929**	**0,0769**	**0,0726**	**0,0722**	**0,0628**	**0,0685**

Table 2: Relative error vs. lower bound (data set 2)

In a second test series (data set 2), we generated 140 data instances with the same dimensions, but with only two or three machines available at each production stage, because this is representative for the situation in the laboratory. The third computational experiment (data set 3) is dedicated to larger data instances. Hence, we generated 80 instances with 50 jobs and $R = 150$. Altogether, we deal with a total of 360 problem instances.

The average relative error

$$\frac{\sum w_j C_j(appr.) - \sum w_j C_j(LB)}{\sum w_j C_j(LB)}$$

of the approximative solution from the lower bound provided by the lp relaxation introduced in Section 2 is displayed in Tables 1-3. To benchmark the different sequencing methods independently, we used them in combination with our list scheduling approach to solve the off-line version of $FFm|r_j, nwt|\sum w_j C_j$ (columns 2-6). The average deviation of the best solution for each instance is given in the next column (best), followed by the result for the hybrid on-line version (on-line). For each problem dimension, we randomly generated 10 instances. To provide further insight, the small instances with 5 or 10 jobs were also solved to optimality. The corresponding results are given in Tables 4 and 5.

The results of the three test series all lead to nearly the same conclusions. We observe that the average performance of the bottleneck-related strategy (2.1) as well as the WSTP sequence (1) is worse than the three bottleneck methods that incorporate the information about preceding production stages (2.2-2.4). Thus, in the case of data set 1, methods (1) and (2.1) yield an average percentage deviation of about 10,4% and 11,4%,

$n \times m$	(1)	(2.1)	(2.2)	(2.3)	(2.4)	best	on-line
50×2	0,0566	0,0642	0,0457	0,0466	0,0472	0,0412	0,0320
50×3	0,0506	0,0582	0,0344	0,0338	0,0306	0,0292	0,0269
50×5	0,1768	0,2012	0,1753	0,1778	0,1779	0,1490	0,1259
50×10	0,2364	0,2636	0,2596	0,2521	0,2488	0,2244	0,2153
total ($\emptyset$)	**0,1301**	**0,1468**	**0,1287**	**0,1276**	**0,1261**	**0,1109**	**0,1000**

Table 3: Relative vs. lower bound (data set 3)

$n \times m$	(1)	(2.1)	(2.2)	(2.3)	(2.4)	best	on-line
5×2	0,0309	0,0344	0,0038	0,0038	0,0038	0,0038	0,0022
5×3	0,0010	0,0117	0,0013	0,0013	0,0013	0,0000	0,0302
10×2	0,0283	0,0323	0,0135	0,0137	0,0070	0,0056	0,0096
10×3	0,0270	0,0272	0,0216	0,0193	0,0156	0,0100	0,0109
10×5	0,0580	0,0505	0,0232	0,0229	0,0210	0,0160	0,0343
10×7	0,0381	0,0414	0,0335	0,0337	0,0256	0,0192	0,0339
total ($\emptyset$)	**0,0306**	**0,0329**	**0,0162**	**0,0158**	**0,0124**	**0,0091**	**0,0202**

Table 4: Relative error vs. optimal solution (data set 1)

$n \times m$	(1)	(2.1)	(2.2)	(2.3)	(2.4)	best	on-line
5×2	0,0088	0,0084	0,0083	0,0020	0,0020	0,0020	0,0115
5×3	0,0180	0,0115	0,0024	0,0021	0,0006	0,0006	0,0033
10×2	0,0403	0,0187	0,0182	0,0161	0,0141	0,0108	0,0096
10×3	0,0421	0,0549	0,0381	0,0381	0,0401	0,0329	0,0331
10×5	0,0088	0,0176	0,0103	0,0102	0,0092	0,0049	0,0109
10×7	0,0240	0,0336	0,0089	0,0071	0,0071	0,0061	0,0206
total ($\emptyset$)	**0,0237**	**0,0241**	**0,0144**	**0,0126**	**0,0122**	**0,0096**	**0,0148**

Table 5: Relative error vs. optimal solution (data set 2)

Data set	# Instances	(1)	(2.1)	(2.2)	(2.3)	(2.4)
1	140	51	35	60	73	80
2	140	54	30	58	73	73
3	80	21	4	24	26	40

Table 6: Number of instances with best objective function value

respectively, whereas methods (2.2)-(2.4) are notably better with a mean relative error of 8,3-9% compared to the lower bound. Naturally, the deviation is significantly smaller in the case of the optimal solution, in detail about 3% for the sequencing methods (1) and (2.1) as well as 1,5% for strategies (2.3)-(2.4). The hybrid approach, i.e., choosing the best job sequence for each instance, leads to an average relative error of 7,6% in the off-line setting as well as 8,1% in the on-line setting. With respect to the optimal solution these values decrease to less than 1% and 2%.

Within the generation of data set 2, we restricted the possible production capacity to two or three machines per stage. Thus, a more balanced workload at the different production stages is implied and consequently, the impact of a single bottleneck stage on the performance of the whole system is weakened. This aspect becomes also apparent in the results obtained for those instances. The arising average deviation is less than the one within data set 1, namely about 6,3% for the hybrid static approach as well as 6,8% for the dynamic setting in comparison with the lower bound. The same is true for the optimal solution where we have a mean error of about 1% and 1,5%, respectively. Not surprisingly, the results for data set 3 with 50 jobs per instance are worse than the other ones. But even an average error of 10% for the on-line method is encouraging in relation to the runtime of the proposed algorithm, which is just a split second. Furthermore, this deviation is achieved in relation to the lower bound, which may be also worse for large instances than for smaller ones.

These results could lead to conjecture that methods (1) and (2.1) were dispensable for a promising hybrid method. To confirm the opposite, take a look at the number of best job sequences yielded by the different sequencing methods shown in Table 6. Sometimes an optimal solution is provided by more than one sequencing method, hence, the sum of the quantities given in one row is greater than the total number of instances. However, within the 51 instances of data set 1 registered for method (1) there are 31, for which the WSTP sequence provides the unique optimum. This ratio is even greater for data sets 2 and 3, namely 39 as well as 19 instances. This shows very clearly that especially method (1) is indispensable for preserving the performance of our hybrid approach. Indeed, the impact of method (2.1) is not that big, but even in this case there are a few instances (set 1: 13, set 2: 4, set 3: 3), for which the best solution is solely provided by method (2.1). In relation to method (1) it is noteworthy that the quantities mentioned above are not evenly spread over the set of instances. In fact, the more processing stages we consider the more "best" job sequences are provided by the WSTP rule. This effect could be caused by the decreasing impact of one (bottleneck) stage on a system with numerous production stages. Hence, especially for instances with five stages and more it is senseful to incorporate at least method (1) beneath the bottleneck approaches (2.2)-(2.4).

Altogether, our results are promising, especially in terms of problem instances closely related to our practical application. Furthermore, our algorithm only requires a split second of computational time to solve all instances, while even the solution of the lp relaxation lasted a few hours for larger instances; not to mention the time needed to solve those instances to optimality. Thus, our method complies with the runtime requirements of our real-life application.

5 Concluding Remarks

This paper is focused on the development of good and extremely fast approximation techniques for solving the $FFm|(r_j), nwt|\sum w_j C_j$-problem with jobs arriving over time to cope with a task arising in steel-producing industries. To the best of our knowledge, this on-line scheduling problem has not been discussed in the research literature before. An algorithm for solving the problem, which meets the run time requirements on the one hand, but also yields reasonable results on the other hand, may exploit the advantages of different fast but crude heuristics. Hence, we decided to develop a hybrid approach. To evaluate the new algorithm we performed a computational study on arbitrarily generated problem instances, which are representative for our practical application. The obtained results are promising.

All findings yielded in this paper are restricted to problems with no waiting time constraints. However, the proposed method can be easily adapted to the case of unlimited intermediate buffer. It would be interesting to evaluate the performance of our algorithm in this context. Furthermore, it might be possible to develop a similar approach for corresponding scheduling problems containing machines of different speed as well as unrelated machines.

References

[1] M. Azizoglu, E. Cakmak, and S. Kondakci. A flexible flowshop problem with total flow time minimization. *European Journal of Operational Research*, 132(3):528–538, 2001.

[2] S. Chakrabarti, C.A. Phillips, A.S. Schulz, D.B. Shmoys, C. Stein, and J. Wein. Improved scheduling algorithms for minsum criteria. *Lecture Notes in Computer Science*, 1099:646–657, 1996.

[3] C. Chekuri and S. Khanna. Approximation algorithms for minimizing average weighted completion time. In J. Y.-T. Leung, editor, *Handbook of Scheduling. Algorithms, Models and Performance Analysis*. Chapman & Hall/CRC, 2004.

[4] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein. Approximation techniques for average completion time scheduling. *SIAM Journal on Computing*, 31(1):146–166, 2001.

[5] M.E. Dyer and L.A. Wolsey. Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics*, 26:255–270, 1990.

[6] M.C. Gomes, A.P. Barbosa-Povoa, and A.Q. Novais. Optimal scheduling for flexible job shop operation. *International Journal of Production Research*, 43(11):2323–2353, 2005.

[7] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

[8] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[9] L.A. Hall, A.S. Schulz, D.B. Shmoys, and J. Wein. Scheduling to minimize average completion time: off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22(3):513–544, 1997.

[10] J. Hurink and S. Knust. List scheduling in a parallel machine environment with precedence constraints and setup times. *Operations Research Letters*, 29:231–239, 2001.

[11] T. Kawaguchi and S. Kyan. Worst case bound of a lrf for the mean weighted flowtime problem. *SIAM Journal on Computing*, 15:1119–1129, 1986.

[12] G.J. Kyparisis and C. Koulamas. A note on weighted completion time minimization in a flexible flow shop. *Operations Research Letters*, 29:5–11, 2001.

[13] R. Linn and W. Zhang. Hybrid flow shop scheduling: A survey. *Computers & Industrial Engineering*, 37:57–61, 1999.

[14] H. Liu, M. Queyranne, and D. Simchi-Levi. On the asymptotic optimality of algorithms for the flow shop problem with release dates. *Naval Research Logistics*, 52:232–242, 2005.

[15] C.D. Paternina-Arboleda, J.R. Montoya-Torres, M.J. Acero-Dominguez, and M.C. Herrera-Hernandez. Scheduling jobs on a k-stage flexible flow-shop. *Annals of Operations Research*, 164(1):29–40, 2008.

[16] C. Phillips, C. Stein, and J. Wein. Scheduling jobs that arrive over time. *Lecture Notes in Computer Science*, 955:86–97, 1995.

[17] C. Phillips, C. Stein, and J. Wein. Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82:199–223, 1998.

[18] M. Pinedo. *Scheduling: Theory, Algorithms and Systems*. Springer, 3rd edition edition, 2008.

[19] K. Pruhs, J. Sgall, and E. Torng. Online scheduling. In J. Y.-T. Leung, editor, *Handbook of Scheduling. Algorithms, Models and Performance Analysis*. Chapman & Hall/CRC, 2004.

[20] M. Queyranne and A.S. Schulz. Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. *SIAM Journal on Computation*, 35(5):1241–1253, 2006.

[21] A.S. Schulz. Scheduling to minimize total weighted completion time: Performance guarantees of lp-based heuristics and lower bounds. *Lecture Notes in Computer Science*, 1084:301–315, 1996.

[22] J. Sgall. On-line scheduling. *Lecture Notes in Computer Science*, 1442:196–231, 1998.

[23] A.P.A. Vestjens. *On-line machine scheduling*. Phd thesis, Institute for Business Engineering and Technology Application - Eindhoven University of Technology, 1997.